# Novint Technologies

## HDAL Software Development Kit

**Revision:** **2.1.3**
**Date:** **Aug. 14, 2008**

**Support:** **Windows XP SP2 (32-bit) and Windows Vista (32-bit)**

# Overview

This SDK contains all the documentation and software development support files needed to develop applications using the Novint Falcon haptic interface device up to and including the Haptic Device Abstraction Layer (HDAL) level. It assumes that the user has already installed the Novint Drivers, including the USB drivers, the DLLs, NOVINT*.BIN, and HDAL.INI files included. This installation includes the setting of NOVINT_DEVICE_SUPPORT and PATH environment variables.

As always, please do not hesitate to contact Novint Technologies Technical Support team with any questions, suggestions or issues. Send email to customersupport@novint.com.

# Major Changes

Major changes in Version 2.1.3 (2.1.2 skipped):

1. Pistol Grip support.
   Earlier versions of the novint*.bin binaries will not recognize a Pistol Grip if it is attached. This version and later is required for Pistol Grip operation.

2. C Compatibility
   hdl.h and hdlu.h have been adjusted to correct some issues with C compatibility. Compatibility with C (instead of just with C++) improves the potential for use with certain scripting languages. The symbol *bool* is now typedef'ed with values *true* and *false* defined. The closing bracket on the extern'd block is now inside a conditional.

3. Fixed bug in repetitious cycling
   If the application went through more than ten iterations of initializing and uninitializing devices, the eleventh initialization would fail. This problem is now fixed.

4. Fixed reinitialization bug
   The bug fix in item 2 above exposed two issues in the recalibration behavior involved when a device is uninitialized and then reinitialized after a USB replug

event. These problems are also fixed. Also, if a device was re-initialized after being closed, the badge color was not correctly set. Finally, a host PC "Restart" operation did not completely recycle the USB ports, leaving the Falcon in an indeterminate state. These are also corrected.

5. Fixed bug in hdlCountDevices
   The count of devices now does not include FTDI devices that are not Falcons.

6. Removed hdlSetConfigPath()
   hdlSetConfigPath() was an unused function (with an incorrect parameter type specification) that had never been removed from the header. Its original use was superseded in the very first version of HDAL, but its declaration was never removed from the header. It is now removed so it cannot be used at all.

7. Type error in HDL_BUILD_VERSION
   The return type of the HDL_BUILD_VERSION function has always been given as _int64. This type is acceptable to Microsoft's IDEs as a substitute for the correct __int64 type, but not to certain other compilers. This unportable substitution was never actually intended and is corrected in this version.

8. Calling convention for ServoOp
   The calling convention specifier "__cdecl" was added to the typedef for the HDLServoOp prototype. On rare occasions, developers may use a default calling convention other than the default __cdecl. This addition provides for such developers.

9. New documentation
   The programmer's guide and the three forms of the API reference have been updated to document these changes.

Major changes in Version 2.1.1:
(Note: 2.1.0 was a limited version only and not released)

1. nullDriver.dll update
   nullDriver.dll was not updated in Version 2.0.0 to contain new entry points required for the new API functions. That problem is now corrected. nullDriver returns 1 for the count of devices and always returns success on hdlInitIndexedDevice(). See the Programmer's Guide, Section 5.4, for an explanation of how nullDriver might be used.

Major changes in Version 2.0.0:

1. LegacySupport folder retired
   The LegacySupport folder provided a means for the user of early production Falcons (so-called T0 and T1 models) to operate. The LegacySupport folder had dhdlc.dll and novint.bin files that were specific to T0 and T1 Falcons. In the new design, dhdlc.dll is common to all Falcons, and the novint.bin files have been replaced with novint_T0.bin and novint_T2.bin. Dhdlc.dll chooses the

appropriate binary file based on the device serial number, which indicates the device type. As device capabilities evolve, it may become necessary to add more binary files to the mix, and this architecture enables seamless transitions as required.

2. New API functions.
    a. int hdlCountDevices()
    This function returns the number of devices connected. Eventually, Novint may add support for non-Falcon devices, and it may or may not be possible to count them. For now, this function counts Falcons.
    b. HDLDeviceHandle HDLAPIENTRY hdlInitIndexedDevice(
    const int index, const char* configPath = 0);
    This function looks and behaves almost like hdlInitNamedDevice(), but instead of a character string identifying the device, an index number identifies the device. From the application's point of view, the specific index isn't particularly important, except that it must be unique from other indexes used in similar calls, and it must be in the range `0 <= n < count`, where `count` is the value returned from hdlCountDevices(). Internally to HDAL, a table of Falcon serial numbers is collected in the hdlCountDevices() call, and the index points into that table. Note that there is no API for returning the actual serial number of the device. That is intentional, since once the device is initialized, it is known by its handle, just as if it had been opened with hdlInitNamedDevice(). See the "multi" example for usage of these two functions.

3. New sample hdal.ini file
Besides the initialization scheme described in (2) above, multiple Falcons can be opened the "traditional" way, by calling hdlInitNamedDevice() with the name of a section in the hdal.ini file. The included hdal.ini file includes two new sections, [FALCON_1] and [FALCON_2], each prepopulated with a real device serial number. To use this file in your own application, make two changes in each section:
    a. Change the name of the section to something meaningful to your app. For instance, if you are associating one Falcon with the left hand and one with the right, you might name the two sections [LEFT] and [RIGHT], and initialize the Falcons with hdlInitNamedDevice("LEFT") and hdlInitNamedDevice("RIGHT"). Of course, if your application uses "Falcon_1" and "Falcon_2" then the sections need not be renamed. Also, new sections can be added; unreferenced sections are ignored.
    b. Change the serial number in the MANUFACTURER_NAME field to match the serial number of the device you want associated with that section. To get the serial number, use FalconTest in the TestUtilities folder, selecting the "Manual" tab. The serial numbers of connected devices are displayed in the list box near the top left of the display.

4. New example
   A new "multi" example has been added. This is a multi-device demo, a very rudimentary implementation of the old gymnasium "Push Ball" exercise. A ball is presented along with some number of colored pushers. The pushers push the ball around on screen. There is no game termination or scoring; it is intended only to demonstrate how multiple Falcons can be used.
   a. How many pushers?
      The number of pushers is determined by a command line argument and the number of connected Falcons. If the command line argument is missing, two Falcons are presumed, and are initialized following the scheme described in (3) above. If less than two Falcons are connected, the demo reports an error and quits. If there is a command line argument, it should be numeric (no error checking; it's a very simple demo). If it is greater than the number of connected Falcons, the demo reports that not enough Falcons are connected, and the game will be played with the number of pushers as there are connected Falcons. If there are enough (or more than enough) Falcons, the game proceeds with the number of pushers requested on the command line.
   b. Associating Falcon to pusher
      A selection and association phase is entered, where the console output asks the user(s) to press any button on the Falcon that should be used to control the pusher of the indicated color. Once all the pushers are associated to a Falcon, the game proceeds. The "Escape" key quits, and the "R" key (case insensitive) resets the ball to the center, to simplify restarting once the ball has been pushed beyond the play area. Remember—it's a programming demo, not a commercial quality game!

5. hdlUninitDevice()
   Under normal circumstances, hdlUninitDevice() behaves as expected. You pass a handle to the device you want to turn off, and that device is now disabled from the running application. However, you need to be aware that "under the covers" hdlUninitDevice() uses a blocking callback function. The Programmer's Guide documents the fact that HDAL currently allows only one blocking callback at a time. This means that you must not call hdlUninitDevice() while another thread is possibly creating a blocking callback function. If that happens, behavior is unpredictable.

6. Improved processor sharing
   This version includes a bug fix that had resulted in applications tending to use a single processor, rather than sharing the workload among what processors are available.

7. Tolerance for other FTDI devices
   Previous versions of HDAL did not properly tolerate the presence of other devices that used FTDI USB interface devices. If other FTDI devices were connected, the Falcon could not be seen by HDAL. This problem is now resolved.

Major changes in Version 1.2.0:

1. Adaptive control of servo tick time. In Version 1.1.0, improved control of servo tick time was introduced. The control depended on available documentation, but later testing showed that some XP machines exhibited undocumented behavior. This version is adaptive, in that at initialization time, the correct timing control is calculated from a sample of actual timing behavior.
2. Reduced time spent in device communication. The previous versions of dhdlc.dll used a communication function described as a "non-overlapped" call, wherein the servo tick thread waited for the command message packet to be clocked out to the Falcon. Beginning in this version, an "overlapped" call is used, which initiates the transfer and returns immediately, freeing up more time in the servo tick for haptic calculations.
3. Correct motor bias calculations in novint.bin. Novint.bin compensates for variation between devices and variation over time in the circuits that control the motors for the production of forces. Under unusual, rarely seen conditions, this compensation could result in a noticeable error on the first game after a power recycling. This error has been corrected in this version.
4. Additional data in servo log file. To demonstrate the effectiveness of (2) above, the servo rate logging feature has been expanded by adding a column of data to the output. To compare the effectiveness of the new communications method, perform the following steps explicitly:
    a. Copy this SDK's hdl.dll and hdal.ini file to a folder containing a game or application executable.
    b. Add the following lines to the [DEFAULT] section of hdal.ini:
       SERVO_LOG=30
       SERVO_LOG_THRESHOLD=0
       The first line will capture 30 seconds of data, about 30,000 lines. If more or less data is desired, set the value to the number of seconds (1,000 lines per second) desired. Note that if you want to analyze the data with Microsoft Excel 2003 or earlier, less than 64,000 lines must be used.
    c. Using NDSSetter, "point" to a previous SDK or driver package (the folder containing bin and config).
    d. Run the game or application for as many seconds as specified in the SERVO_LOG in hdal.ini.
    e. Repeat steps (c) and (d), pointing NDSSetter to this SDK.
    f. Compare the "SynchTime" data in the two servoLog* files produced in step d.
5. Add include and lib subfolders to LegacySupport. This facilitates the build process by developers who use the NOVINT_DEVICE_SUPPORT environment variable in their project build files to reference the headers and lib file.
    a. If you are using a very early Falcon ("T0" or "T1"), you will need to run NDSSetter and navigate to the LegacySupport folder in either this package, in the SDK version 1.1.0 package, or in the Falcon Drivers folder of the normal user's installation folder. Users of later Falcons ("T2" and later) should point to the "normal" folder.

    b. As a reminder, one option available to application developers is to use the NOVINT_DEVICE_SUPPORT environment variable to reference the folder holding the header files and the folder holding the lib files. For instance, in Visual Studio, in the C/C++ section of the project properties, you might consider setting the Additional Include Directories to include $(NOVINT_DEVICE_SUPPORT)\include instead of a hardcoded path. Using this setup allows the NDSSetter utility to ensure that development paths and execution paths are coherent. Eventually, new features will be added that will appear in the include files and lib files; when this happens, it will be important to use build files that are consistent with execution files.
6. Correct notes in API Reference on search order for configuration file.
7. NDSSetter now makes environment variable HDAL_BASE track NOVINT_DEVICE_SUPPORT, if it exists; if it does not exist, it is not created.

Major changes in Version 1.1.0:

1. Improvements in haptic device initialization. Previously, in rare circumstances, the Falcon required manual resetting to recover from an unsynchronized state. Manual intervention is no longer required.
2. Improvements in servo rate stability. Deviation from the nominal 1 msec servo interval has been reduced, resulting in smoother haptic simulation.
3. Enhanced servo tick rate logging for diagnostics.
4. Added LegacySupport folder to accommodate developers with very early Falcons. Unless you have been specifically directed to do so, you will not need to use this folder. If you do, just use NDSSetter to point to it instead of to the top level folder.

Major changes in Version 1.0.1:

1. This version is identical to Version 1.0.0 except that it corrects an error in novint.bin. If a user had used NDSSetter to point the NOVINT_DEVICE_SUPPORT and PATH environment variables to reference this SDK instead of the normal Falcon device drivers, and then the Falcon went through a reset cycle, the incorrect version of novint.bin would have been downloaded and the Falcon would fail to initialize. The evidence for this would be that the badge would not go through the normal red/blue stages, and the Falcon would feel very sluggish and non-responsive.

   We recommend that Version 1.0.0 of this SDK be uninstalled and this one used in its place. If the user has modified the contents of that version, such as modifying the examples, then a "quick fix" would be to copy novint.bin from this version to the 1.0.0 version.

Major changes in Version 1.0.0:

1. Initial release.

# Usage

1. Install the SDK to a folder of your choice.
2. After installing, run NDSSetter to point to the new installation folder. Note that this requires that PATH and NOVINT_DEVICE_SUPPORT environment variables must have been previously established, which is done when Falcon Drivers are initially installed.
3. Choose one of the Examples to build and test. The examples use relative addressing for includes and libraries, so no environment variable setup is needed.
4. For your own projects, use your company's standards for how to locate the include and lib folders. Common options include:
    a. Set an environment variable to point to the HDAL folder, and use this environment variable in each project's properties.
    b. Set the IDE's global properties to reference the HDAL/include and HDAL/lib folders
    c. Set each projects's properties to reference the HDAL/include and HDAL/lib folders.

    Option (a) has a distinct advantage that will become most apparent when the next version of the SDK becomes available. You will be able to retain old versions of this SDK for comparison purposes by the simple expedient of changing a single environment variable.

# Contents of the Distribution

The distribution is provided as a self-contained installer. It includes the following folders:

- bin
  - the DLLs and NOVINT.BIN files needed at runtime
- config
  - HDAL.INI
- doc
  - reference documentation for the HDAL API, in PDF, HTML, and CHM formats
  - a Programmer's Guide document to help you get started
- examples
  - a "basic test" program the implements a "hello, world" level of haptic programming for OpenGL, DX9, VC 6, VC 8, and 3D Game Studio.
- include
  - the hdl and hdlu header files required for compiling HDAL applications
- lib
  - the hdl.lib file required for linking HDAL applications
- utilities
  - NDSSetter, a utility that lets you switch DLL configurations by browsing to folders, rather than doing a lot of typing

The bin and config folders are basically copies of the contents of the driver installation you should already have. They are provided here to give you some control over versions. The previous drivers are updated under control of NVeNT, Novint's automatic updater system. When you run NVeNT to play Novint's games, you may get a new driver from time to time. As a software developer, you may want to be in full control of when your drivers change. Having this pair of folders gives you that control. Running NDSSetter in the utilities folder allows you to easily switch between different versions.